Graphic User Interfaces
Introduction

Modern computer operating systems include presentation services that provide the user with a GUI (Graphic User Interface). As a result, computer users now demand application programs that also provide GUIs. Fortunately, the Java platform provides extensive support for building them. GUIs display in graphical form on the screen. To program a GUI, you must master event-driven programming because GUI-oriented programs are event-driven.

In this module, you will learn how to use the most common user interface components in the Java Swing package.

Objectives

- To use inheritance to customize frames
- To understand how user-interface components are added to a container
- To understand the use of layout managers to arrange user interface components in a container
- To become familiar with common user interface components, such as buttons, combo boxes, text areas and menus
- To build programs that handle events from user interface components
- To learn how to browse the Java documentation

Terminology of a GUI

All interactive output and input should pass through your program’s GUI. Therefore, the GUI consists of what the user sees on the screen and code to process user actions such as clicking the mouse or typing on the keyboard. Your program must properly relate those actions to the elements displayed in the GUI and perform the appropriate activities in response.

Components comprise a major part of a GUI. In the Java platform, components are predefined standard elements such as buttons, text fields, frame windows, and dialog boxes. The Swing and AWT APIs provide a large repertoire of components.

The display space on the screen is also a component. A Java application window is a frame window. Frame windows have a title and a border; buttons for closing, minimizing, and maximizing the window; and can contain a menu bar. All your controls (such as check boxes, labels, text areas, etc.) need to be put into the frame to display them. Components like the frame and panels are also called containers.
Using Inheritance to Customize Frames

For building a complex GUI with many interface components added to a frame, you usually start by inheriting the JFrame class. The JFrame (belongs to the javax.swing) class defined and inherits many useful fields and methods. The most commonly used methods are, setSize and setVisible.

Layout management

Components are arranged by placing them inside containers. Each container has a layout manager that directs the arrangement of its components. Three useful layout managers are the border layout, the flow layout and the grid layout.

Border layout – Groups the container into five areas: centre, north, west, south and east. When adding a component to a container with the border layout, specify the NORTH, EAST, SOUTH, WEST or CENTRE position to specify where the component lies in relation to other components.

Read and evaluate the following example.

```java
import java.awt.*;
import javax.swing.*;

public class TestLayout extends JFrame {
    private static final int FRAME_WIDTH=200;
    private static final int FRAME_HEIGHT=200;

    JPanel p;
    JButton up, down, left, right;

    public TestLayout(String title) { //constructor the GUI
        //initialize the fields here
        p = new JPanel();
        up = new JButton("UP");
        down = new JButton("DOWN");
        left = new JButton("LEFT");
        right = new JButton("RIGHT");

        // set panel layout
        p.setLayout(new BorderLayout());

        // add components to the panel
        p.add(up, BorderLayout.NORTH);
        p.add(down, BorderLayout.SOUTH);
        p.add(left, BorderLayout.WEST);
        p.add(right, BorderLayout.EAST);

        //add to frame and set frame
        add(p);
        setTitle(title);
        setSize(FRAME_WIDTH, FRAME_HEIGHT);
        setVisible(true);
    }

    public static void main(String[] args) {
        new TestLayout("TestLayout");
    }
}
```
**Flow layout** – A flow layout simply arranges its components from left to right and starts a new row when there is no more room in the current row. A JPanel uses a flow layout by default.

```java
p.setLayout(new FlowLayout());
JButton A = new JButton("ACT");
JButton B = new JButton("QLD");
JButton C = new JButton("NSW");
JButton D = new JButton("VIC");
JButton E = new JButton("TAS");
JButton F = new JButton("NT");
JButton G = new JButton("WA");
p.add(A);
p.add(B);
p.add(C);
p.add(D);
p.add(E);
p.add(F);
p.add(G);
```

**Grid layout** – The grid layout arranges components in a grid with a fixed number of rows and columns, resizing each of the components so that they all have the same size.

```java
p.setLayout(new GridLayout(4,3));
JButton n1 = new JButton("1");
JButton n2 = new JButton("2");
JButton n3 = new JButton("3");
JButton n4 = new JButton("4");
JButton n5 = new JButton("5");
JButton n6 = new JButton("6");
JButton n7 = new JButton("7");
JButton n8 = new JButton("8");
JButton n9 = new JButton("9");
JButton n0 = new JButton("0");
JButton ndot = new JButton(".");
JButton nce = new JButton("CE");
p.add(n7);
p.add(n8);
p.add(n9);
p.add(n4);
p.add(n5);
p.add(n6);
p.add(n1);
p.add(n2);
p.add(n3);
p.add(n0);
p.add(ndot);
p.add(nce);
```

The above three layout managers are most commonly used layouts. You can create acceptable-looking layouts in nearly all situations by nesting panels. You give each panel an appropriate layout manager. Panels don’t have visible borders, so you can use as many panels as you need to organize your components.
**Choices (Radio Buttons, Check Boxes and Combo boxes)**

Radio buttons work like the station selector buttons on a car radio: If you select a new station, the old station is automatically deselected. You add radio buttons into a ButtonGroup so that only one button in the group is on at any time. The following screen shot is an example of RadioButtons and its code.

```
import java.awt.*;
import javax.swing.*;

public class TestRadioButton extends JFrame
{
    // declare the fields here
    private static final int FRAME_WIDTH=400;
    private static final int FRAME_HEIGHT=100;

    JPanel p;
    JRadioButton n1, n2, n3;
    ButtonGroup g;

    public TestRadioButton(String title)  // constructor GUI
    {
        // initialize all fields here
        p = new JPanel();
        n1 = new JRadioButton("Small");
        n2 = new JRadioButton("Medium");
        n3 = new JRadioButton("Large");

        // ButtonGroup to hold buttons
        g = new ButtonGroup();
        // add buttons into group
        g.add(n1);
        g.add(n2);
        g.add(n3);

        // set panel layout
        p.setLayout(new GridLayout(1,3));

        // add components to the panel
        p.add(n1);
        p.add(n2);
        p.add(n3);

        // add to frame and set frame
        add(p);
        setTitle(title);
        setSize(FRAME_WIDTH, FRAME_HEIGHT);
        setVisible(true);
    }

    public static void main(String[] args)
    {
        new TestRadioButton("Radio Button");
    }
}
```
Note, you cannot add a button group to a panel directly. A panel has to have each individual radio button added on to it.

A **Check box** is a user interface component with two states: checked and unchecked. You use a group of check boxes when one selection does not exclude another. That means, a user can select more than one value. The following screen shot is a check box and its corresponding code pieces.

```java
p.setLayout(new GridLayout(1,3));
JCheckBox n1 = new JCheckBox("Rocky");
JCheckBox n2 = new JCheckBox("Mackay");
JCheckBox n3 = new JCheckBox("Bundy");
JCheckBox n4 = new JCheckBox("Glandston");
p.add(n1);
p.add(n2);
p.add(n3);
p.add(n4);
```

Note: you do not need to group check boxes. Each individual check box can be directly added to the panel.

A **Combo Box** is a drop down selection list. To create a combo box and add items to the list, use:

```java
JComboBox state = new JComboBox();
state.addItem("NSW");
state.addItem("QLD");
state.addItem("VIC");
```

Note: you can add ComboBox into a panel, e.g.

```java
JPanel p = new JPanel();
p.add(state);
```

Radio buttons, check boxes, and combo boxes generate action events, just as buttons do. You need to add action listeners and implement `actionPerformed` method when necessary. The programs `FontViewer.java` and `FontViewerFrame.java` (page 794, 795 of the text) demonstrate how to use these three components in a GUI. Note if many components share a listener, this inner listener class can be put into the class constructor of outside any other methods. If a listener is particular applied for one component, it would be better to put the listener class close to the component definition.
Activity
Heading “How To 18.1” on pages 799, 800, 801, Laying Out a User Interface, is very important for a GUI programming. It is helpful for you to start your assignment. You are advised to read through this part carefully, and apply the 5 steps listed in this section in your assignment.

Menus
A frame contains a menu bar. The menu bar contains menus. A menu contains submenus and menu items. Only menu items generate action events. The sequence normally used is to:
  • create a menu bar container (to hold menus)
  • create a menu container for each menu (to hold menu items)
  • create menu items and handle events for each menu item
When your menu has more levels, the code can become very long.
The FontViewer2.java and FontViewer2Frame.java are a complete menu example. Make sure you understand this problem.

Reading
Text book:
  Chapter 18: Graphic User Interface (whole chapter required)

Review questions
Review exercises:
  Page 816: Exercise R18.2, R18.3, R18.9, R18.10, R18.12,
Programming exercises:
  Practise panel overlay exercises in Lab session and
  Start working on your assignment.

References
**Lab session**

1) **Panel overlay technique**

One of the annoying things about using most of the available layout managers in a simple fashion is the fact that all of the components placed on the panel use up as much area as the largest component. So, for instance, if we place a simple button and a text area onto a panel using the FlowLayout manager, the button takes up just as much room as the text area - even though the text area may be very large.

Another annoying feature is the difficulty in placing different numbers of components beside one another, or above one another. For instance, what if we want to place a text field above two buttons that are side-by-side? There are layout managers that will let you do that (e.g. the GridBagLayout or the BoxLayout), but they aren't all that easy to understand and use; and, in the case of the BoxLayout, some features are deprecated. There must be an easier way - and there is (within limits).

Consider the following diagram:

My Text Field

<table>
<thead>
<tr>
<th>My Text Field</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Button 1</td>
</tr>
<tr>
<td>Button 3</td>
</tr>
</tbody>
</table>

How could this layout be achieved using the simple layout managers? One way would be to use multiple panels to contain the various components.

Place buttons 1 and 2 in a panel (call it panelOneTwo) using a GridLayout. Do the same with buttons 3 and 4 by placing them in another similar panel (call it panelThreeFour).

For consistency of coding, we could place the text field in another panel on its own. It wouldn't really matter what layout we used. Call it panelText.

Now we place panelOneTwo and panelThreeFour in another panel, beside one another. We could use a Flowlayout or a GridLayout. Probably a GridLayout would be best. We'll call this panel the buttonPanel.

Now we place the panelText and the buttonPanel into another panel that we might call the desktopPanel.

It is the desktopPanel that we place into the mother frame!

It requires a bit of planning and forethought. Most layouts can be solved using this multi-panel technique. The best approach is to draw your required layout onto a piece of paper, and then work out how to build up your simple panels, layer by layer, to achieve your goal. You may not always get exactly what you want; but you can usually get very close to the desired effect.
2) After reading the panel overlay technique document, write a program to implement the layout described in 1) by extending JFrame class. The following java swing components are needed. Note an object instance (variable name) is different from its Caption.

**JButton**: for Button 1, Button 2, Button 3 and Button 4

**JTextField**: for MyTextField

**JPanel**: for destopPanel, panelText, buttonPanel, panelOneTwo and panelThreeFour

**JFrame**: for the main frame to hold desktopPanel

3) Once you finished 1) and 2), add an actionListener class to output a message in MyTextField area to indicate which of the four buttons is pressed.

4) This is a menu creating exercise. The following code is an example to create two menus (File and Help) onto a GUI. Each menu only contains one menu item in the code. Inspect this code and add one more menu item to the File menu, say add a “New Game” item, when the “New Game” menu item is selected, program call a method called reset(). In reset() method, pop up a message “A New Game is starting”.

```java
import javax.swing.*;
import java.awt.event.*;
public class MenuExercise extends JFrame
{
    private static final int FRAME_WIDTH = 700;
    private static final int FRAME_HEIGHT = 500;
    public static void main(String[] args)
    { Jframe frame = new MenuExercise();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
        frame.setVisible(true);
        frame.setResizable(false); }
    public MenuExercise () { JMenuBar menuBar = new JMenuBar();
        setJMenuBar(menuBar);  // method inherited from java.awt.Frame
        menuBar.add(createFileMenu());
        menuBar.add(createHelpMenu());
    }
    public JMenu createFileMenu() { JMenu menu = new JMenu("File");
        menu.add(createExitItem());
        return menu; }
    public JMenuItem createExitItem() { JMenuItem item = new JMenuItem("Exit");
        ActionListener listener = new MenuItemListener();
        item.addActionListener(listener);
        return item; }
    public JMenuItem createExitItem() { JMenuItem item = new JMenuItem("Exit");
        class MenuItemListener implements ActionListener
        { public void actionPerformed(ActionEvent event) { if (JOptionPane.showConfirmDialog(null, "DO YOU REALLY WANT TO EXIT??", "",JOptionPane.YES_NO_OPTION, JOptionPane.WARNING_MESSAGE) == 0) System.exit(0); } }
        ActionListener listener = new MenuItemListener();
    }
```
item.addActionListener(listener);
return item;
}  
//end of create File Menu

//Create Help Menu

public JMenu createHelpMenu()
{
    JMenu menu= new JMenu("Help");
    menu.add(createAboutItem());
    return menu;
}

public JMenuItem createAboutItem()
{
   JMenuItem item = new JMenuItem("About");
class MenuItemListener implements ActionListener
   {
      public void actionPerformed(ActionEvent event)
      {
         JOptionPane.showMessageDialog(null,"This is a menu exercise!");
      }
   }
   ActionListener listener = new MenuItemListener();
   item.addActionListener(listener);
   return item;
}

// end of create Help Menu
// end of menu creating

5) Exercise P18.1 and P18.2, write an application with three buttons labelled “Red”, “Green”, and “Blue” that changes the background colour of a panel in the centre of the frame to red, green and blue. Add icons (can be any image icon) to the button of “Red”, “Green” and “Blue”.

6) Simulate a traffic light application (see following image) by using a frame containing 3 buttons in Red, Yellow and Green colours. Then add a Timer object to change the traffic colour every 10 seconds (you may use the template in next page).

7) Add your digital clock from week 4’s lab to your traffic light application (as in the following figure).
// import java classes from library

public class TrafficLightFrame extends JFrame
{
    private JPanel mainPanel;
    private JButton topButton;
    private JButton middleButton;
    private JButton bottomButton;

    private static final int FRAME_WIDTH = 100;
    private static final int FRAME_HEIGHT = 300;

    public TrafficLightFrame()
    {
        // create mainPanel, topButton, middleButton and bottomButton objects

        //add buttons to the panel

        // add main panel to the frame

        // inner class to implement ActionListener    (decide how to change colours)

        //create a listener object
        // create a Timer object
        // start timer;
    }

    // other methods if needed

    public static void main(String[] args)
    {
        JFrame frame = new TrafficLightFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
        frame.setVisible(true);
    }
}